CAPÍTULO 3

CONCORRÊNCIA

Prezado(a) aluno(a),

Neste capítulo você estudará assuntos fundamentais para o entendimento da concorrência em sistemas multiprogramáveis, tais como: interrupção, tipos de E/S, Buffering, Spooling e Reentrância.

O bom entendimento do mecanismo de funcionamento da interrupção, bem como seus diversos tipos: interrupção de hardware, de exceção e de software, é essencial para a compreensão dos capítulos futuros. Dê bastante atenção às interrupções, pois é uma das principais evoluções responsáveis pela multiprogramação.

Bom estudo!

Fala Professor

3.1. Introdução

Em sistemas monoprogramáveis, somente um programa podia estar em execução por vez e, por isso, o processador permanecia dedicado exclusivamente a uma única tarefa. Se um programa fizesse alguma operação de entrada e saída (E/S), o processador ficaria ocioso durante toda a realização da operação. Esse tempo de espera é relativamente longo, já que as operações de E/S são muito lentas, se comparadas com a velocidade com que o processador executa as instruções. Em sistemas monoprogramáveis, os recursos computacionais como processador, memória e dispositivos de E/S são utilizados de maneira pouco eficiente, limitando o desempenho. Muitas vezes, esses recursos permanecem ociosos durante longos períodos de tempo, provocando desperdício da capacidade computacional. Assim, devido às limitações dos sistemas monoprogramáveis surgiram os sistemas multiprogramáveis.

Os sistemas multiprogramáveis tornam mais eficiente à utilização dos recursos computacionais, por permitirem a execução simultânea (concorrente) de vários programas. Nesses sistemas, quando um programa está realizando uma operação de E/S, outros podem utilizar o processador. A possibilidade de o processador executar instruções em paralelo com as operações de E/S, permite que diversas tarefas sejam executadas

ao mesmo tempo. Outro aspecto a se ressaltar é a utilização da memória principal, que, ao invés de ficar subutilizada (como em sistemas monoprogramáveis), pode conter vários programas residentes concorrendo pela utilização do processador. Com isso, há um melhor aproveitamento dos recursos computacionais em sistemas multiprogramáveis.

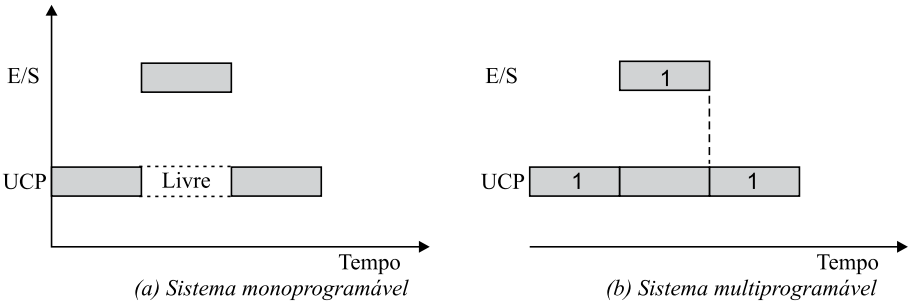

Figura 3-11: Sistema monoprogramável x sistema multiprogramável
Fonte: [1] – Machado e Maia, 2004. Adaptação.

3.2. Interrupção

Durante a execução de um programa, alguns eventos inesperados podem ocorrer, interrompendo o seu fluxo normal de execução e ocasionando o desvio forçado. Tais eventos são conhecidos por interrupção, mecanismo básico e fundamental para o funcionamento dos sistemas operacionais multiprogramáveis.

Sempre que ocorre uma interrupção, o sistema operacional é chamado para executar a rotina de tratamento. Antes de realizar o desvio, o sistema operacional deve salvar o estado do processo interrompido, para que este possa continuar do ponto onde parou quando voltar a executar. Existem três tipos de interrupção: interrupção de hardware, de exceção e interrupção de software [4].

Fala Professor

Alguns autores não fazem essa distinção entre os tipos de interrupções. Sempre que o termo interrupção for empregado sozinho, estará se referindo às interrupções de hardware.

A **interrupção de hardware** ocorre em consequência da sinalização de algum dispositivo de hardware externo ao processador. Geralmente são os dispositivos de E/S, enviando sinais sobre as operações de E/S, mas também pode ser gerada pelo temporizador (*timer*).

Conceitos

Exemplos de dispositivos que geram interrupção de hardware: mouse, teclado, dispositivo de E/S, avisando ao processador que alguma operação de E/S terminou e, o temporizador, avisando que o tempo (*time slice*) de um processo terminou.

A **exceção** ocorre em consequência de algum erro durante a execução de uma instrução do próprio programa. Há exceções que sinalizam a ocorrência de algum erro aritmético e outras que sinalizam erros relacionados à segurança (proteção) do sistema.

Conceitos

São exemplos de exceções aritméticas: divisão por zero, *overflow*, *underflow*. São exemplos de exceções de segurança: tentativa de acesso ilegal à memória, estouro de pilha, tentativa de execução de instrução privilegiada no modo usuário.

A interrupção de software é, na verdade, uma chamada de sistema, realizada sempre quando um utilitário ou aplicativo solicita recursos de hardware ao sistema operacional (as chamadas de sistemas serão vistas mais a frente).

Conceitos

São exemplos de chamadas de sistema: solicitação leitura ou gravação de arquivo, leitura do teclado, impressão na tela, etc.

3.2.1. Mecanismo de interrupção

Ao final da execução de cada instrução, a unidade de controle verifica se ocorreu algum tipo de interrupção. Em caso afirmativo, a execução do programa é interrompida e o fluxo de controle é desviado para alguma rotina responsável por tratar o evento ocorrido, chamado de **rotina de tratamento de interrupção**. Para que o programa possa voltar a ser executado posteriormente, é necessário que as informações sobre o estado do programa sejam salvas imediatamente após a ocorrência da interrupção. Tais informações consistem no conteúdo dos registradores e são denominadas de **contexto de execução.** Após o tratamento da interrupção, o contexto de execução deverá ser restaurado para que o programa continue executando do ponto onde parou – como se nada tivesse acontecido.

Para cada tipo de interrupção existe uma rotina de tratamento associada, para onde o fluxo de execução deverá ser desviado. Para obter o endereço da rotina de tratamento da interrupção ocorrida, é necessário identificar a origem do evento. Uma das formas de realizar isso é por meio de uma estrutura chamada vetor de interrupção, que contém o endereço inicial de todas as rotinas de tratamento existentes, associadas a cada tipo de evento. Toda interrupção é identificada por um número e esse número é usado para consultar o vetor de interrupção e obter o endereço inicial da rotina de tratamento.

Assim, quando ocorre uma interrupção, o conteúdo dos registradores (contexto de execução) é salvo na pilha de controle; em seguida, a origem do evento é identificada e é obtido o endereço da rotina de tratamento. A rotina de tratamento é então, executada. Ao final, o conteúdo dos registradores é restaurado e o programa volta a executar do ponto onde parou.

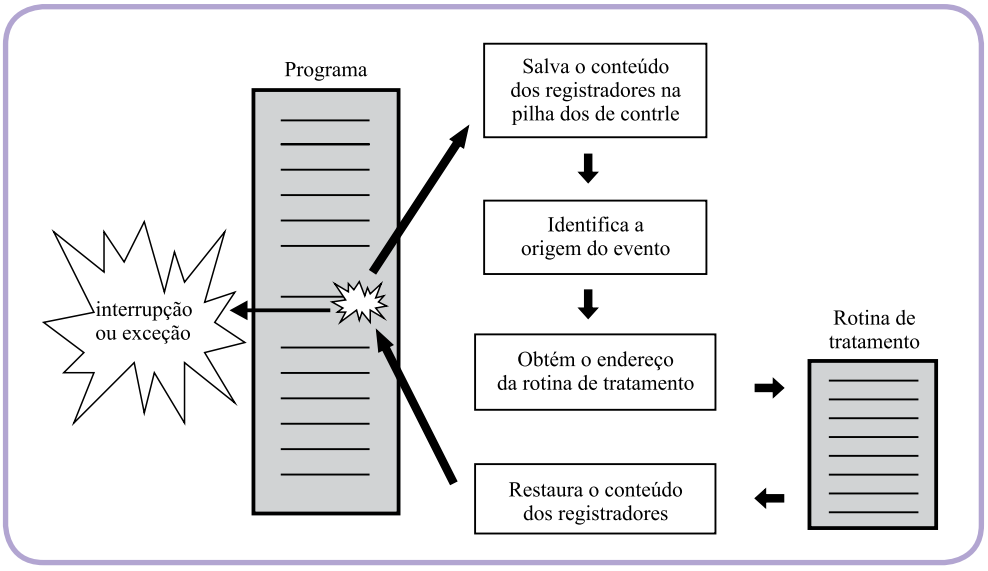

Figura 3-12: Mecanismo de interrupção
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Os eventos que causam a interrupção podem ser classificados como síncronos ou assíncronos. Um evento é dito **síncrono,** quando é resultado direto da execução do programa concorrente. Nesse caso, se um mesmo programa for executado várias vezes com a mesma entrada de dados, os eventos síncronos ocorrerão sempre nos mesmos pontos (instruções) do programa (exemplo: exceção de divisão por zero, *overflow, underflow*, etc). Um evento é classificado como **assíncrono,** quando é independente dos dados de entrada e das instruções do programa, ou seja, quando pode ocorrer em qualquer ponto do programa (exemplo: interrupção gerada pelo mouse, teclado, periféricos, etc). Assim, as **interrupções de hardware** são classificadas como **assíncronas**, pois independem das instruções e da entrada de dados do programa. Já as **exceções** e as **interrupções de software** (chamadas de sistema) são classificadas como **síncronas**, pois sempre ocorrerão nos mesmos pontos do programa, se tiverem a mesma entrada de dados.

As interrupções de hardware, por serem imprevisíveis (decorrentes de eventos assíncronos), podem ocorrer múltiplas vezes, como no caso de vários dispositivos de E/S informarem ao processador o término de uma operação de E/S. Para tratar apropriadamente esse caso, as demais interrupções estarão desabilitadas durante a execução da rotina de tratamento. Em outras palavras, elas só serão tratadas quando a rotina de tratamento da interrupção atual terminar. Interrupções com a característica de poder ser desabilitada são chamadas **mascaráveis**. Em função da importância da interrupção, cada uma possui uma prioridade, utilizada para definir a ordem de atendimento das interrupções simultâneas.

As interrupções que não podem ser desabilitadas são chamadas **não-mascaráveis**. Alguns exemplos de eventos que geram interrupções não-mascaráveis são: pressionar o botão de reset, falha no hardware (memória, processador, etc).

3.3. Operações de entrada/saída

Nos primeiros sistemas de computação, os periféricos eram controlados pelo processador por meio de instruções especiais, chamadas instruções de E/S. Essas instruções continham detalhes específicos de cada periférico (por exemplo, em qual trilha e setor um bloco de dados deve ser lido ou gravado no disco-rígido). Devido a isso, esse modelo criava uma forte dependência entre o processador e os dispositivos de E/S.

Para evitar esse problema, criou-se o **controlador** ou **interface,** que permitiu ao processador operar de maneira independente dos dispositivos de E/S. Com esse novo elemento, o processador não precisava mais se

comunicar diretamente com os periféricos, nem conhecer detalhes específicos de operação dos periféricos. Ao invés disso, ele se comunicava apenas com o controlador, responsável por conhecer os detalhes de operação específicos de cada periférico. Isso simplificou bastante as instruções de E/S do processador.

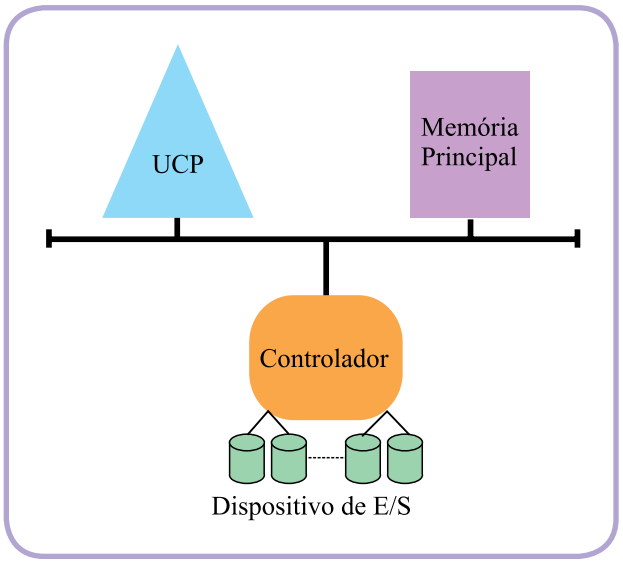

Figura 3-13: Controlador dos dispositivos de E/S
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Por meio do controlador, o processador gerenciava as operações de E/S, sem se preocupar com os detalhes de implementação de cada dispositivo.

3.3.1. E/S programada

Um dos primeiros modelos de operação de E/S, foi a E/S programada. Neste modelo, após o processador iniciar a transferência de dados, ficava consultando o estado do periférico sucessivamente, até que a operação de E/S chegasse ao fim. Esse controle, chamado **E/S programada**, mantinha o processador ocupado até o término da E/S (espera ocupada, do inglês *busy wait*). Como o processador executa uma instrução muito mais rapidamente que uma operação de E/S realizada pelo controlador, havia um enorme desperdício de tempo do processador.

3.3.2. E/S por *Polling*

A evolução do modelo anterior foi permitir que algumas instruções pudessem ser executadas entre sucessivas consultas sobre o estado de uma operação de E/S. Essa forma de E/S foi chamada ***Polling*** e introduziu um certo grau de paralelismo. Isso porque outros programas poderiam executar, enquanto uma operação de E/S era realizada; mas, em determinados intervalos de tempo, o sistema operacional deveria interrompê-los, para verificar o estado da operação de E/S. Apesar de ser um

modelo melhor que o anterior, caso houvesse várias operações de E/S pendentes, o processamento seria interrompido diversas vezes para a verificação do estado das operações. A maior dificuldade desse modelo consistia em determinar a frequência para a realização do *polling*.

3.3.3. E/S Controlada por interrupção

Com a implementação do mecanismo de interrupção, as operações de E/S puderam ser realizadas de forma mais eficiente. Ao invés de o sistema ficar periodicamente testando o estado das operações pendentes, o próprio controlador interrompe o processador para informar o término da operação. Nesse mecanismo, chamado de **E/S controlada por interrupção**, o processador, após um comando de leitura ou gravação, permanece livre para o processamento de outras tarefas. O controlador, por sua vez, fica responsável pela operação de E/S. No entanto, quando essa operação termina, o controlador interrompe o processador para que este realize a transferência de dados para a memória principal. Após essa transferência, o processador está livre para executar outros programas. Esse modelo revela-se bastante eficiente, quando comparado com os anteriores. Apesar disso, se houver transferência de um grande volume de dados, o processador será interrompido diversas vezes, reduzindo o desempenho. Para solucionar esse problema, foi implementada uma técnica de transferência de dados chamada DMA (*Direct Memory Access*).

3.3.4. Acesso direto à memória (DMA)

Conceitos

A técnica de **DMA (*Direct Memory Access*)** permite que o controlador de E/S transmita um bloco de dados entre os dispositivos de E/S e a memória principal.

O controlador acessa a memória diretamente, sem a necessidade da intervenção do processador, exceto no início e no final da transferência. Por exemplo, quando o sistema deseja ler ou gravar um arquivo, o processador informa ao controlador o dispositivo de E/S, a localização, a posição inicial da memória onde os dados serão lidos ou gravados e o tamanho do arquivo. Após passar tais informações ao controlador, o processador fica livre para executar outros programas. O controlador realiza a operação de E/S, bem como a transferência de dados entre a memória e o dispositivo de E/S, e, somente

ao final, interrompe o processador para avisar que a operação foi concluída. A área de memória utilizada pelo controlador na técnica de DMA é chamada *buffer de entrada/saída*.

3.4. Buffering

Conceitos

A técnica de ***Buffering*** consiste na utilização de uma área na memória principal, denominada *buffer*, para a transferência de dados entre os dispositivos de E/S e a memória.

O objetivo do *Buffering* é minimizar a disparidade entre as velocidades do processador e dispositivos de E/S. Isso é alcançado por buscar, a maior parte do tempo, manter ocupados o processador e os dispositivos de E/S.

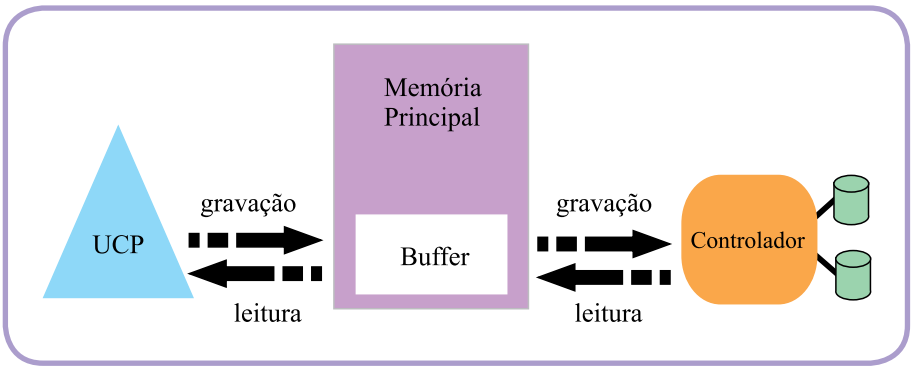

Figura 3-14: Operações de E/S utilizando Buffering
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Por exemplo, em uma operação de gravação, os dados a serem gravados são colocados no Buffer e o processador é imediatamente liberado para execução de outros programas. Enquanto isso, o controlador traz os dados do buffer para que sejam gravados pelo dispositivo de E/S. Assim, o processador consegue trabalhar em paralelo com os dispositivos de E/S, aumentando o desempenho do sistema.

3.5. Spooling

A técnica de ***Spooling*** (***Simultaneos Peripheral Operation On-Line***) foi introduzida o final da década de 1950, para aumentar o grau de concor-

rência e a eficiência dos sistemas operacionais. Semelhante à técnica de *buffering*, a técnica de *spooling* utiliza uma área em disco, como se fosse um grande buffer. Ela serve para controlar o acesso a dispositivos que atendem apenas a uma requisição por vez. Atualmente esta técnica está presente na maioria dos sistemas operacionais e é utilizada no gerenciamento da impressão.

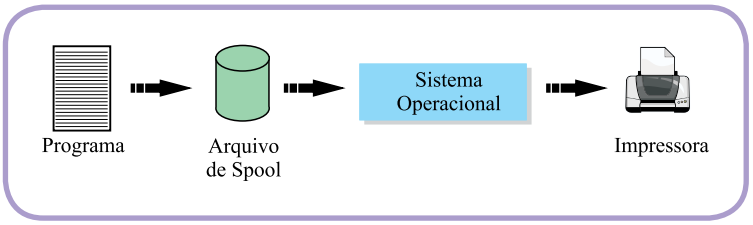

Figura 3-15: Técnica de Spooling
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Quando um programa necessita imprimir um arquivo, ele envia um comando ao gerenciador de impressão. As informações a serem impressas são gravadas em um arquivo em disco, conhecido como arquivo de *spool*, liberando imediatamente o programa para outras atividades. Em seguida, o sistema operacional fica encarregado de direcionar o conteúdo do arquivo de *spool* para a impressora. Assim, o programa que está imprimindo executa em paralelo com a impressão, aumentando a produtividade do sistema.

3.6. Reentrância

Quando o *loader* carrega um programa da memória secundária para a principal, ele aloca áreas de código, dados e pilha. Em sistemas multiprogramáveis, são comuns vários usuários utilizarem o mesmo aplicativo simultaneamente. Assim, serão alocadas áreas diferentes de código, dados e pilha, para cada aplicação de usuário. Porém, quando alguns usuários utilizarem a mesma aplicação, haverá diferentes áreas de código alocadas que possuem o mesmo conteúdo (mesmo programa executável), causando desperdício da memória. Nesse caso, seria desejável que a área de código de cada aplicação fosse alocada uma única vez e compartilhada com os usuários, reduzindo a utilização da memória.

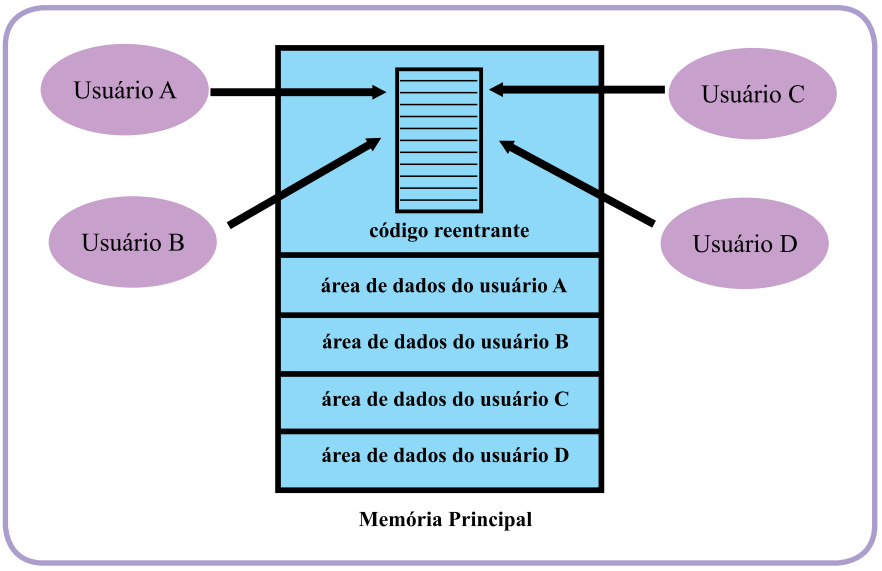

Figura 3-16: Reentrância
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Conceitos

Reentrância é a capacidade de um código executável (código reentrante) ser compartilhado por vários usuários, exigindo que apenas uma cópia do programa esteja na memória.

Em geral, os utilitários do sistema, como editores de texto, compiladores e *linkers*, são de código reentrante, proporcionando uma utilização mais eficiente da utilização da memória. Alguns sistemas operacionais permitem a implementação do conceito de reentrância por aplicações desenvolvidas pelos próprios usuários.

Indicações

[1] MACHADO, F.B. e MAIA, L.P. ***Arquitetura de Sistemas Operacionais***. 4.ed. LTC, 2007.

[2] SILBERSCHATZ, A., GALVIN, P.B., GAGNE, G. ***Fundamentos de Sistemas Operacionais***. 6.ed. LTC, 2004.

[3]TANENBAUM, A.S. ***Sistemas Operacionais Modernos***. 2.ed. Pearson Brasil, 2007.

[4] OLIVEIRA, R.S., CARISSIMI, A.S., TOSCANI, S.S. ***Sistemas Operacionais***. 3.ed. Sagra-Luzzato. 2004.

Conceitos

Atividades

1. Pesquise o que é concorrência e diga como esse conceito está presente nos sistemas operacionais multiprogramáveis?
2. Por que o mecanismo de interrupção é fundamental para a implementação da multiprogramação?
3. Explique o mecanismo de funcionamento das interrupções.
4. O que são eventos síncronos e assíncronos? Como esses eventos estão relacionados ao mecanismo de (i) interrupção de hardware, (ii) interrupção de software e (iii) exceção?
5. Dê exemplos de eventos associados ao mecanismo de exceção.
6. Qual a vantagem da E/S controlada por interrupção, comparada com a técnica de *polling*?
7. O que é DMA e qual a vantagem desta técnica?
8. Como a técnica de *buffering* permite aumentar a concorrência em um sistema computacional?
9. Explique o mecanismo de *spooling* de impressão.
10. Em um sistema multiprogramável, seus usuários utilizam o mesmo editor de textos (200 Kb), compilador (300 Kb), software de correio eletrônico (200 Kb) e uma aplicação corporativa (500 Kb). Caso o sistema não implemente reentrância, qual o espaço de memória principal ocupado pelos programas, quando 10 usuários estiverem utilizando todas as aplicações simultaneamente? Qual o espaço liberado, quando o sistema implementa reentrância em todas as aplicações?
11. Por que a questão da proteção torna-se fundamental em ambientes multiprogramáveis?